INKA USER'S GUIDE

KIRILL LAPIDUS

Contents

1. Introduction 2
2. **Histogrammer** 3
2.1. Basic usage 3
2.2. Multiple file input 3
2.3. Arbitrary cuts 4
2.4. Secondary vertex cuts 4
2.5. Integral spectrum 4
2.6. Additional methods 4
3. **Fitter** 5
3.1. Basic usage 5
3.2. Changing the initial fit parameters 5
3.3. Reading/writing fit parameters from/into a file 6
3.4. Changing fit parameters for particular bins 6
3.5. Integral spectrum 6
4. **Corrector** 8
4.1. Basic usage 8
4.2. Multiple simulation files 8
4.3. Cuts 8
4.4. Employing the Geant info number 8

Date: January 26, 2012.

1. Introduction

INKA is a class library that allows to make a differential analysis via invariant mass (for example, $K^0_S \to \pi^+\pi^-$ or $\Lambda \to p + \pi^-$)[1]. It includes three main classes: `Histogrammer`, `Fitter` and `Corrector`.

The main feature of INKA is that a user doesn't have to create/fill/save any single histogram. Even in the case INKA is not suited for a particular analysis, you might try the nifty `Histogrammer` class.

[1]There is a way to use INKA for a single-particle analysis, please contact the author if you need more information.

2. **Histogrammer**

The task of this class is to produce histograms. As an input it takes a file (or files) after the final analysis.

2.1. **Basic usage.** Let's assume that you need to produce invariant mass distributions in 11 bins in y (rapidity) variable and 14 bins in p_t (transverse momentum). Then you would use `Histogrammer` like this:

```
//Create a Histogrammer object
//each histogram will have 250 bins in the region (0,1000) MeV
Histogrammer *hist = new Histogrammer(250,0,1000);

//Set input and output files
hist->set_input_output("/Users/klapidus/development/pNb35_NT_all_201011.root",
                       "/Users/klapidus/development/hist_output.root");

//Specify the name of the tree (ntuple) in the input file
hist->init("K0SNT");

//add variables (names of branches) which you'd like to study,
//specify numbers of bins and ranges
hist->add_variable("PtK0S",14,0.,1050.,"YK0S",11,0.36788,1.46788);

hist->execute(0); //0 -- all events in the input file
hist->finalize();
```

Note, that you should provide `Histogrammer` with the names of the variables in the tree (branches); in the considered example they are `PtK0S` and `YK0S`.

Assume you'd like to make additionally an (m_t, y)-analysis in the same time. Then you add the following command to the script above:

```
hist->add_variable("MtK0S",15,500.,1500.,"YK0S",11,0.36788,1.46788);
```

`Histogrammer` also supports the one-dimensional case. This way you make a p_t-analysis:

```
hist->add_variable("PtK0S",14,0.,1050.);
```

Now all the necessary histograms are in the output file. What you usually need to do next is to fit them in order to find background contributions and extract signals. This is a job of `Fitter` (see section 3).

2.2. **Multiple file input.** In order to use several files as an input, one should use the method `Histogrammer::add_file`. Note, that in this case one has also to use the method `Histogrammer::set_output`. An example of usage:

```
Histogrammer *hist = new Histogrammer(250,0,1000);
//Set the input and output files
hist->set_output("/Users/klapidus/development/hist_output.root");
hist->add_file("/Users/klapidus/development/input1.root");
hist->add_file("/Users/klapidus/development/input2.root");
hist->add_file("/Users/klapidus/development/input2.root");
...
```

2.3. **Arbitrary cuts.** To apply cuts to the data sample, use this method:

`Histogrammer::add_cut(TString name, TString operator, Float_t value);`

Let's consider an example: you'd like to apply a cut on the `ThetaK0S` variable: `ThetaK0S < 30`. In the same time, you'd like to fulfill the conditions `N_Tracks > 3` and `N_Tracks < 7`. The following code will do the job:

```
Histogrammer *hist = new Histogrammer(250,0,1000);
...
hist->add_cut("ThetaK0S","<", 30); //add the first cut
hist->add_cut("N_Tracks",">", 3);  //add the second cut
hist->add_cut("N_Tracks","<", 7);  //add the third cut
...
hist->execute(0); //0 -- all events in the input file
```

All cuts you add in this way will be applied to all histograms, including the integral spectrum (see below).

2.4. **Secondary vertex cuts.** To apply secondary vertex cuts there is a special method:

```
Histogrammer::set_svertex_cuts(Float_t aVerDistX, Float_t aMinTrackDist,
                               Float_t aVerDistA, Float_t aVerDistB)
```

These cuts will be also applied to all histograms, including the integral spectrum (see below).

2.5. **Integral spectrum.** It is useful to have an opportunity to produce an integral spectrum, without any restrictions (except for the vertex or other added cuts). A special method exists for this case:

`Histogrammer::set_inv_mass_all(nbins, min, max)`

`nbins` — number of bins you'd like to have; `min (max)` — minimum (maximum) value of the variable.

If you use this method, then a folder named `integral_spectrum` will be created in the output root-file.

2.6. **Additional methods.** To set the title of the X-axis:

`Histogrammer::set_invmass_title(TString ainvmass_title)`

3. Fitter

3.1. **Basic usage.** In case you don't need any special features, work with `Fitter` is simple: you only need to specify the input file and which histograms you'd like to fit.

```
//create a Fitter object
//250 is a minimum number of counts in the histogram
//380. and 600. define a range in which the fits will be performed
Fitter *fit = new Fitter(250, 380., 600.);
//input file (output of the Histogrammer)
fit->set_input("/Users/klapidus/development/hist_output.root");
fit->init();
fit->add_variable("PtK0S",14,0.,1050.,"YK0S", 11, 0.36788, 1.46788);
fit->execute();
fit->finalize();
```

After `Fitter` is done in the input file a folder named `fit_PtK0S_YK0S` (for the considered example) with raw spectra will appear.

3.2. **Changing the initial fit parameters.** For the fitting of histograms the following 12-parameter function is used[2]:

```
y = par0*( TMath::Gaus(x,par1,par2) ) + //gauss1
    par3*( TMath::Gaus(x,par1,par4) ) + //gauss2
    par5 + par6*x + par7*x*x + par8*x*x*x + //polynomial
    par9*( TMath::Landau(x,par10,par11) );  //Landau function
```

Two gaussians correspond to a signal, a background is fitted by a polynomial + Landau function.

Initial values of parameters and their limits are set inside the code (`fitter.cc`). There is a possibility to change them (globally, for all fits) in a user program with the following methods:

```
Fitter::set_fit_param(Int_t n_fit_param, Float_t value)
Fitter::set_fit_param_limits(Int_t n_fit_param, Float_t value_min, Float_t value_max)
```

An example of usage:

```
Fitter *fit = new Fitter(250);
...
fit->set_fit_param(10, 525.);
fit->set_fit_param(11, 30.);
fit->set_fit_param_limits(10, 521., 529.);
...
fit->execute();
fit->finalize();
```

[2]At the moment only this function can be used. Eventually, more flexibility will be offered.

3.3. **Reading/writing fit parameters from/into a file.** Use this method to write all obtained fit parameters into a file:

```
Fitter::write_fit_param(TString avarname1, TString avarname2, TString aoutput);
```

and this one to read fit parameters from a file (this way you *set them as initial values for the fit*):

```
Fitter::read_fit_param(TString avarname1, Int_t anbins1,
                       TString avarname2, Int_t anbins2, TString ainput);
```

An example of usage:

```
Fitter *fit = new Fitter(...);
...
fit->read_fit_param("PtK0S",nptbins,"YK0S",nybins,"param_output1.txt");
fit->write_fit_param("PtK0S","YK0S","param_output1.txt");
...
```

Note that a file with fit parameters also contains fit ranges (parameters `xleft` and `xright`) for every single fit, so you can modify it as well.

3.4. **Changing fit parameters for particular bins.** *This method is not available anymore, since the needed functionality is completely covered by parameter input/output (see 3.3).* Since the fit doesn't always work perfectly, sometimes it is needed to change the fit parameters for a particular bin. This is possible with the method

```
Fitter::change_fit(Int_t j, Int_t i, Float_t axleft, Float_t axright)
```

where `j` and `i` define a bin. At the moment only the range of the fit can be changed, but it is straightforward to extend the method. In future a more general and flexible approach will be developed.

3.5. **Integral spectrum.** In 2.5 the way to produce an integral spectrum was explained.

With the method

```
Fitter::fit_integral_spectrum(Float_t afit_range_left, Float_t afit_range_right);
```

you can fit the integral spectrum (two parameters define the fitting range) and with

```
Fitter::get_sig_bg_integral_spectrum(Float_t& signal, Float_t& bg)
```

extract the values of the signal and the background in the integral spectrum.

In case you need to access normalized χ^2 (χ^2/NDF) of the fit, there is a method

```
Float_t Fitter::get_chi2_norm_integral_spectrum()
```

Example of usage:

```
...
chi2 = fit->get_chi2_norm_integral_spectrum()
...
```

A method

```
Fitter::get_fit_param_integral_spectrum(Int_t n_fit_param, Float_t& value)
```

allows to access the value of a parameter after the fit of the integral spectrum.

4. Corrector

4.1. **Basic usage.** `Corrector` requires three files: 1) a file with raw distributions (delivered by `Fitter`), 2) a file with simulated (usually in 4π) particles (all necessary variables, like `PtK0S`, `YK0S` should be in a tree), and 3) a file after final analysis (symmetric to the experimental data).

```
//create a Corrector object
Corrector *corr = new Corrector();
//give a name of the tree
corr->init("K0SNT");
//set the input file with raw histograms
corr->set_raw_file("/Users/klapidus/development/hist_output.root");
//set an output file
corr->set_output("/Users/klapidus/development/corr_output.root");
//add a geant-cleaner file and a file after analysis
//"geant-cleaner" is a file with initial simulated particles
corr->add_file("/Users/klapidus/development/K0S_simDst_clean_shift.1.2.root",
               "/Users/klapidus/development/pNb_K0S_simMicroDst_list_shift.root");
//set vertex cuts
corr->set_svertex_cuts(vdx, mtd, vda, vdb);
//add distributions you want to correct
corr->add_variable("PtK0S",nptbins,0.,1050.,"YK0S", nybins, yleft, yright);
corr->execute(0);
corr->correct();
```

4.2. **Multiple simulation files.** `Corrector` supports a possibility to use many input files with weights:

```
corr->add_file("simDst_clean_1.root", "channel1_final_analysis.root", 1.5);
corr->add_file("simDst_clean_2.root", "channel2_final_analysis.root", 7.6);
```

Note, that in case you don't provide a weight, a default value of 1.0 will be assigned.

4.3. **Cuts.** In full analogy with `Histogrammer`, `Corrector` has methods to apply secondary vertex and arbitrary cuts:

```
Corrector::set_svertex_cuts(Float_t aVerDistX, Float_t aMinTrackDist,
                            Float_t aVerDistA, Float_t aVerDistB);
Corrector::add_cut(TString name, TString oper, Float_t value);
```

4.4. **Employing the Geant info number.** To weight entries, which have a Geant info number assigned, the following method should be used:

```
Corrector::add_channel_weight(Int_t geant_info_num, Float_t weight);
```

An example of usage:

```
Corrector *corr = new Corrector();
...
corr->add_channel_weight(3227810, 0.3325);
corr->add_channel_weight(1835783, 0.00147);
...
```

```
corr->correct();
```